

# ReactJS.

The Component model

# Topics

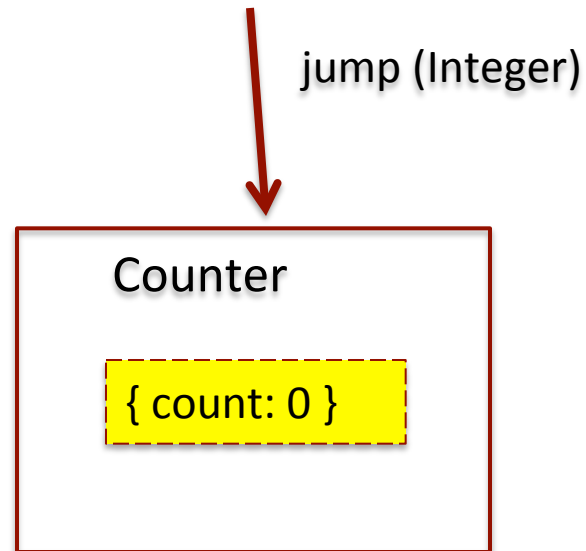
- **Component State**
  - **Basis for dynamic, interactive UI**
- **The Virtual DOM**
- **Data Flow patterns**
- **Lifecycle methods**

# Component DATA

- **Two sources of data for a component:**
  1. **Props - Immutable; Passed in to a component.**
  2. ***State* – Dynamic; Managed internally by the component**
    - **\*\*\* The basis for dynamic and interactive Uis \*\*\***
- **Props-related component features:**
  - Set default prop values.
  - Type-checking.
- **State-related component features:**
  - Initialize state values.
  - Change the state values- the `setState()` method.
    - Change some or all state values (merge, not overwrite).
    - **\*\*\* Automatically causes component to re-render. \*\*\***

# Component State - Example

- The Counter component.
- Ref. 06\_state.js
- Component class coding features:
  1. Custom functions, e.g. incrementCount().
  2. Static class property, e.g. defaultProps.
  3. Class instance property, e.g. state.



# React's event system.

- Cross-browser support.
- Event handlers receive `SyntheticEvent` – a cross-browser wrapper for the browser's native event.
  - Same interface as native event
- Event naming convention slightly different from native:

React	Native
onClick	onclick
onChange	onchange
onSubmit	onsubmit

- See <https://reactjs.org/docs/events.html> for full details,

# Re-rendering

- EX.: The Counter component.

*User clicks 'increment' button*

*→ onClick event handler (incrementCounter) executed*

*→ state is changed (setState())*

*→ render() method executed*

# Modifying the DOM

- **Traditional performance best practice:**
  1. **Avoid expensive DOM operations.**
  2. **Minimize access to the DOM.**
  3. **Update elements offline before reinserting into the DOM.**
  4. **Avoid tweaking layouts in Javascript.**
- **Should the developer be responsible for low-level DOM optimization? Probably not.**
  - **React solution is the Virtual DOM.**
  - **A challenge to established thinking!**

# The Virtual DOM

- **Re-render everything on every update.**
  - Sounds expensive!
- **How?**
  1. Create lightweight description of app's UI (The Virtual DOM)
  2. Perform *diff* operation between it and the previous (virtual) UI state.
  3. Compute the minimal set of changes to apply to (real) DOM.
  4. Batch execute all updates to real DOM.
- **Benefits:**
  - a) Clean – Clean, descriptive programming model
  - b) Fast - Optimized DOM updates and reflows.



# Re-rendering (detail)

- **EX.: The Counter component.**

*User clicks 'increment' button*

*→ onClick event handler (incrementCounter) executed*

*→ state is changed (setState())*

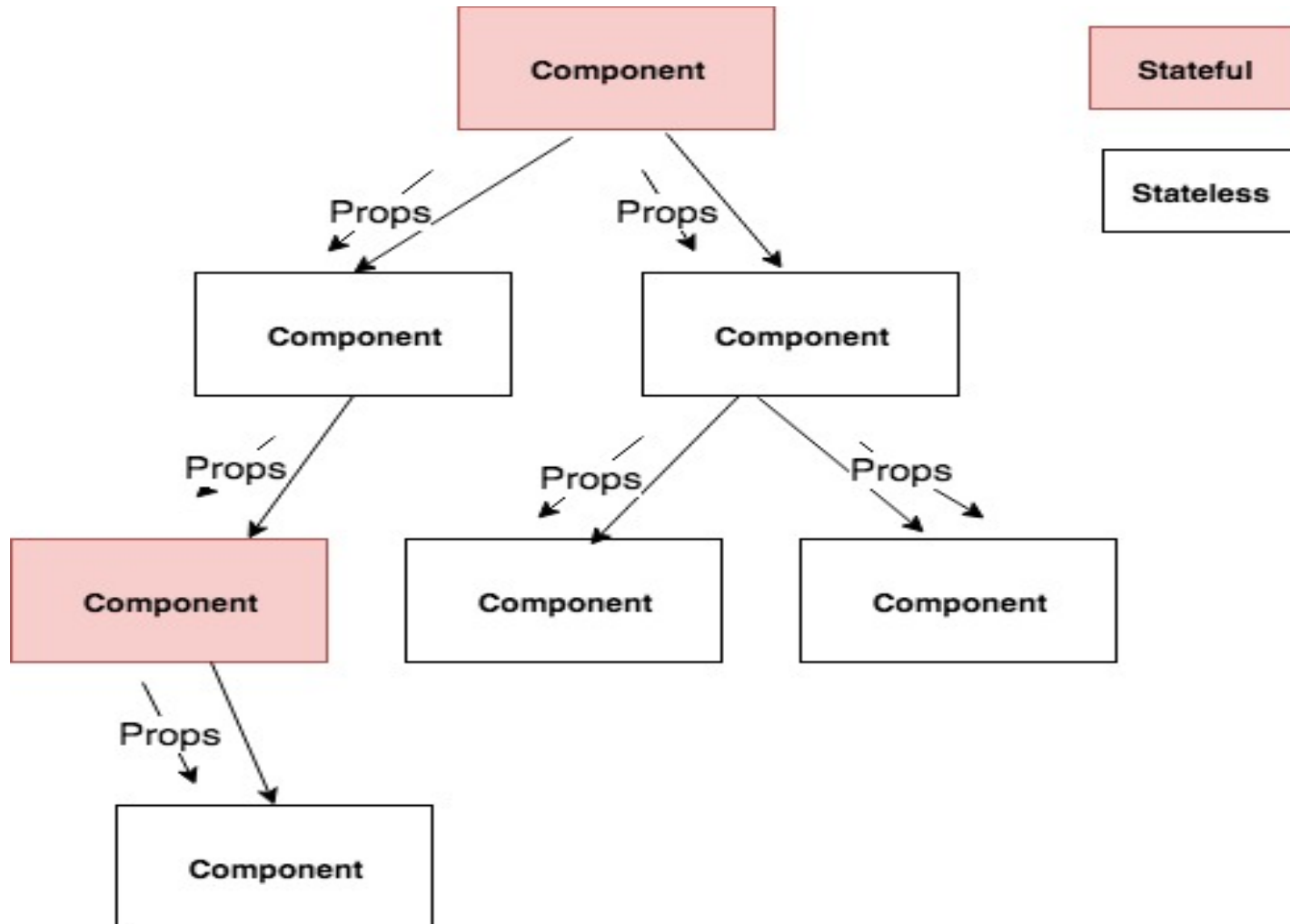
*→ render() method executed*

*→ The Virtual DOM has changed*

*→ React diffs the changes (between the current and previous Virtual DOM)*

*→ React batch updates the Real DOM*

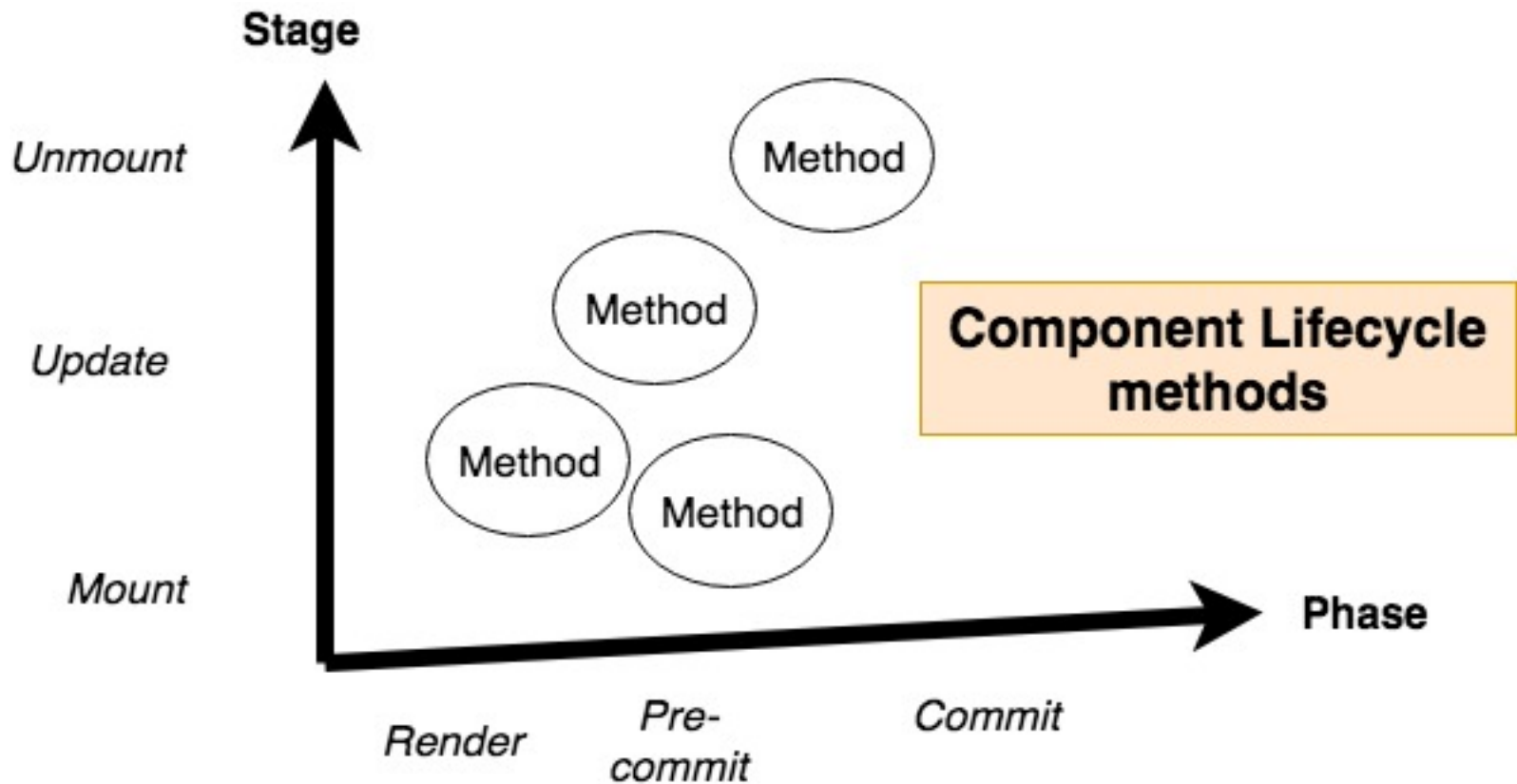
# Unidirectional data flow



# Unidirectional data flow

- In a React app, data flows uni-directionally **ONLY**.
  - Most other SPA frameworks use two-way data binding.
- In a multi-component app, a common pattern is:
  - A small subset (maybe only 1) of components will be statefull – the majority will be stateless.
  - Statefull components:
    - Pass state changes to subordinate components via props, If necessary.
    - Calls `setState()` to update its state.
      - *React guarantees subordinate components are re-rendered with new prop values.*

# Component *Lifecycle methods*



# Component *Lifecycle* methods

- **Methods invoked by React at specific times in a component's lifecycle (Most are optional).**
- **Lifecycle stages:**
  - 1 **Mounting (Initialization).**
  - 2 **Update.**
    - a) **New props.**
    - b) **setState();.**
    - c) **forceUpdate.**
  - 3 **Un-mounting.**
- **Phases:**
  - **Render phase.**
  - **Pre-commit phase (Pre DOM update)**
  - **Commit phase (Post DOM update)**
  - **.**

# *The Lifecycle* methods

- **shouldComponentUpdate()** – returns boolean – can cause a component to skip re-rendering.
- **getDerivedStateFromProps()** - when a component state object is computed from its prop values.
- **componentDidUpdate()** – executed after a rendering has updated real DOM; used to perform real DOM manipulations, e.g. set up external subscription or cause side-effect.
- **componentDidMount()** – executed once, after component has mounted (see later)
- **componentWillUnmount()**; executed before a component is about to unmount; Perform cleanup operations, e.g. remove external subscription.

# *The Lifecycle* methods

- *shouldComponentUpdate()* ✓
- *getDerivedStateFromProps()*
- *render()*. ✓
- *componentDidUpdate()*
- *componentDidMount()* ✓
- *componentWillUnmount()*

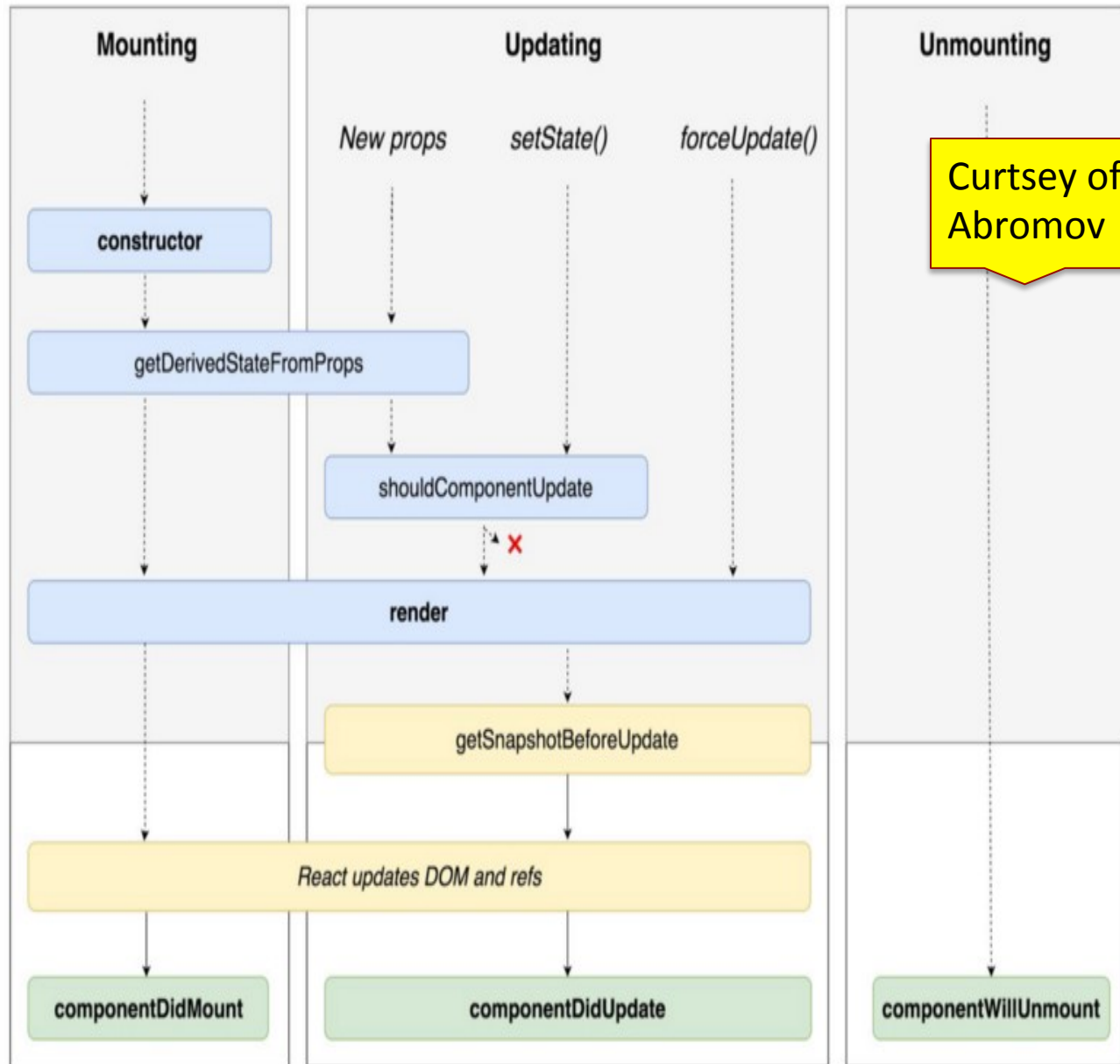
**"Render Phase"**  
 Pure and has no side effects.  
 May be paused, aborted or  
 restarted by React.

**"Pre-Commit Phase"**

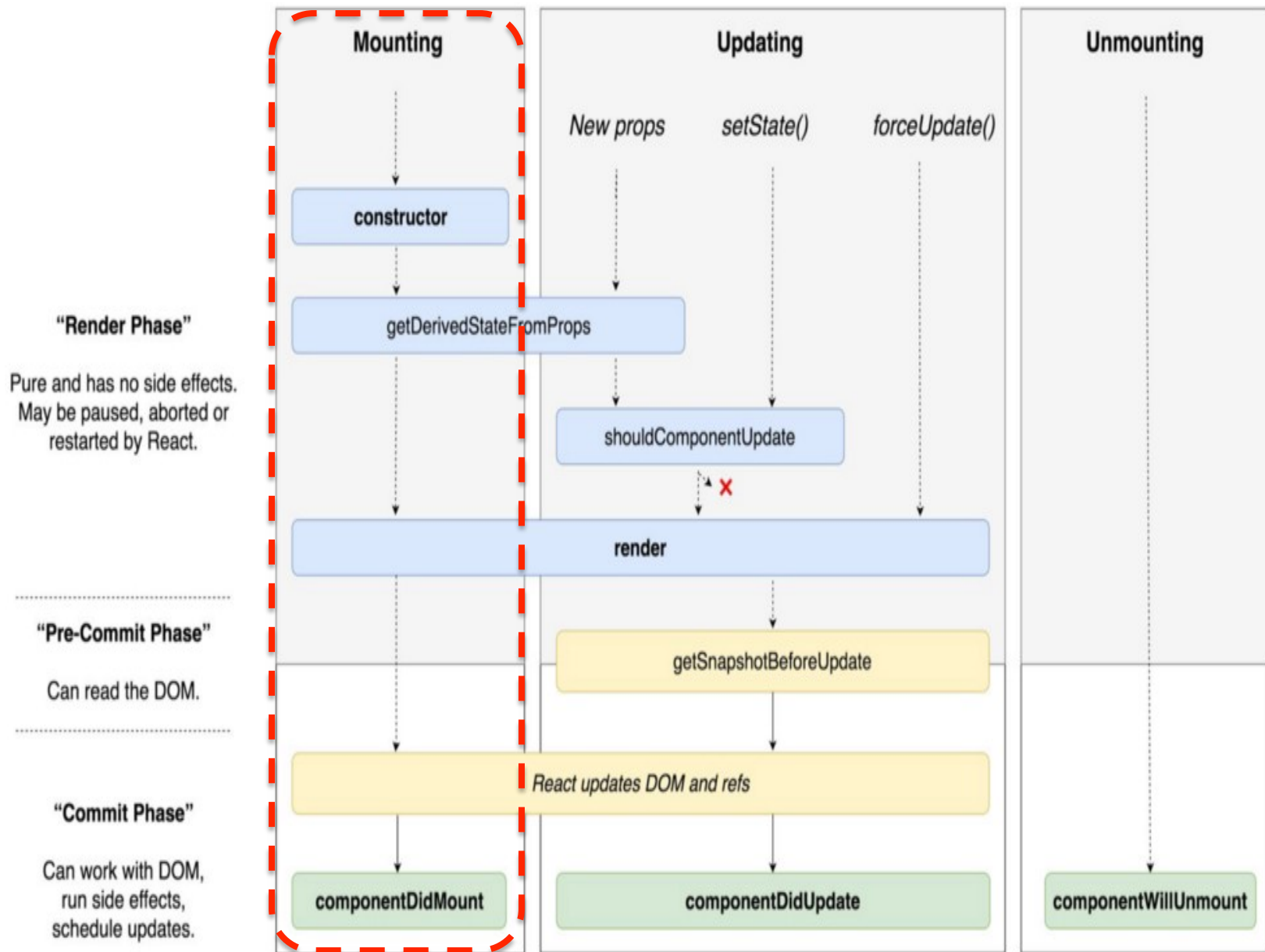
Can read the DOM.

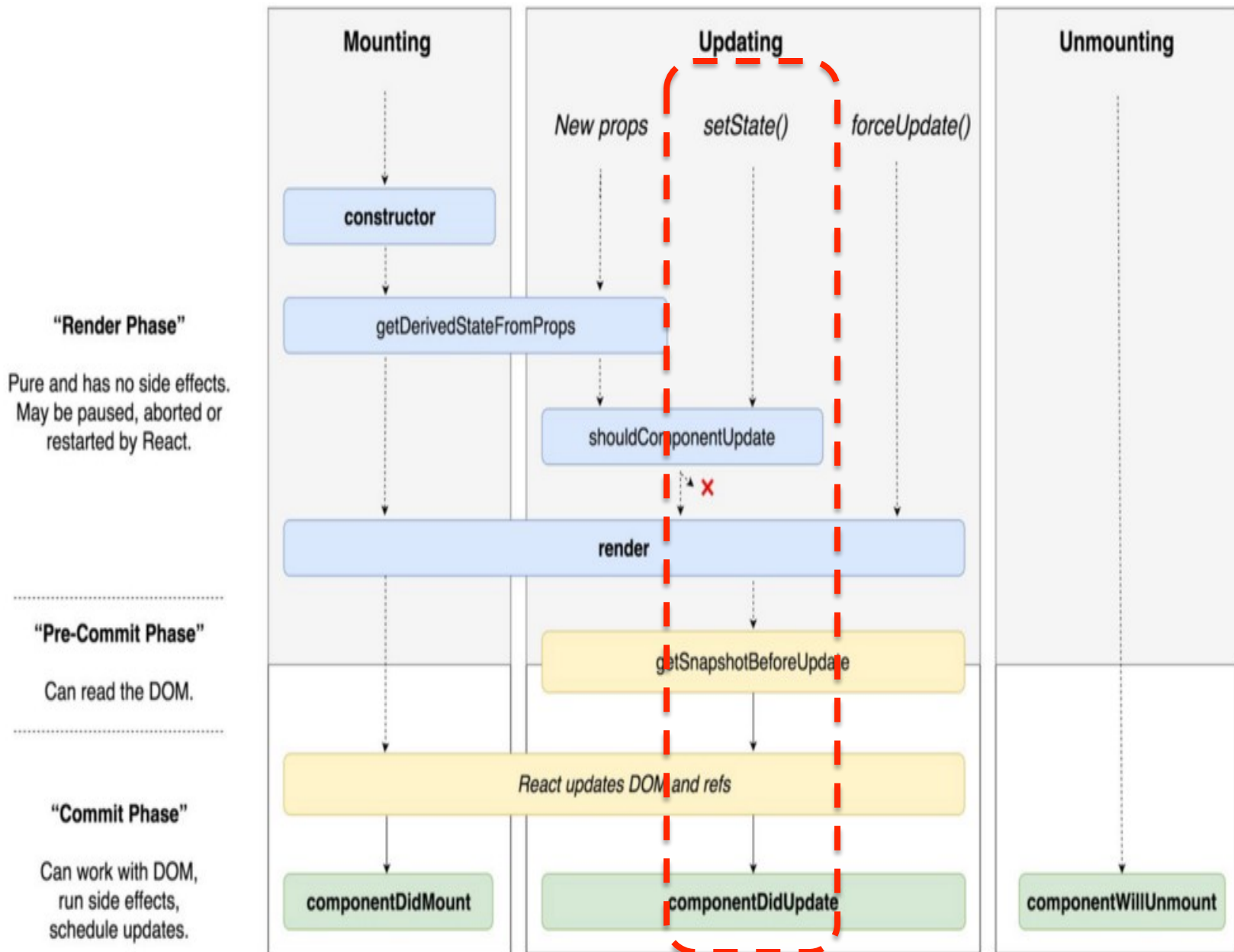
**"Commit Phase"**

Can work with DOM,  
 run side effects,  
 schedule updates.



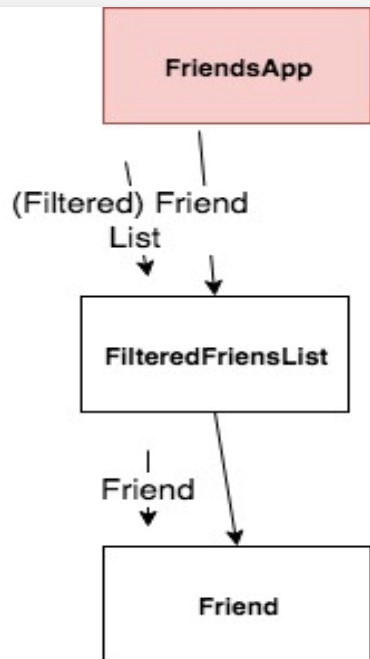






# Sample App

Component  
hierarchy  
diagram



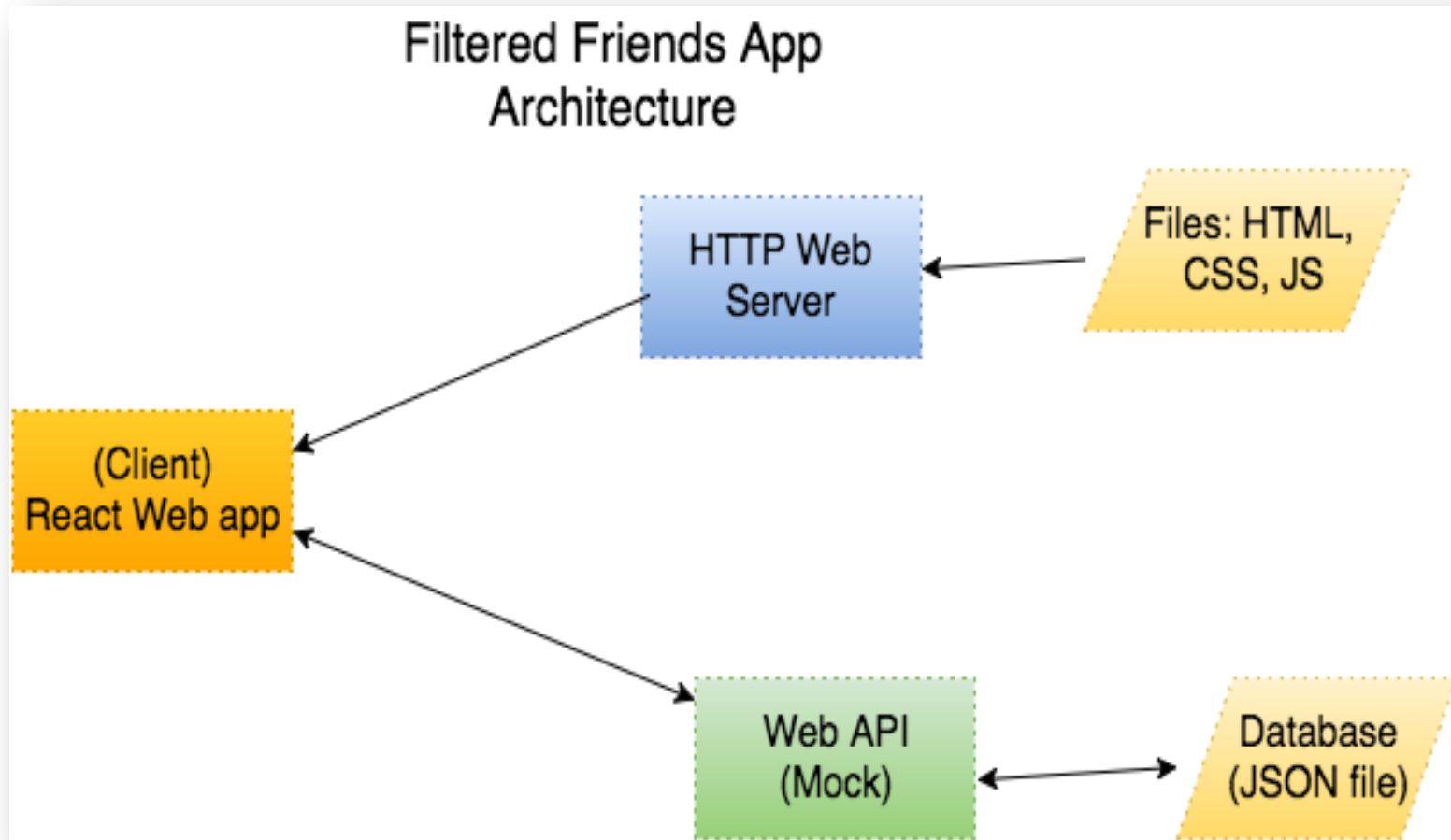
*FriendsApp* component:

1. Manages app's state (i.e. text box content).
2. Computes matching friends list.
3. Controls list re-rendering.

## Friends List

- **Joe Bloggs**  
[jbloggs@here.com](mailto:jbloggs@here.com)
- **Paula Smith**  
[psmith@here.com](mailto:psmith@here.com)
- **Catherine Dwyer**  
[cdwyer@here.com](mailto:cdwyer@here.com)
- **Paul Briggs**  
[pbriggs@here.com](mailto:pbriggs@here.com)

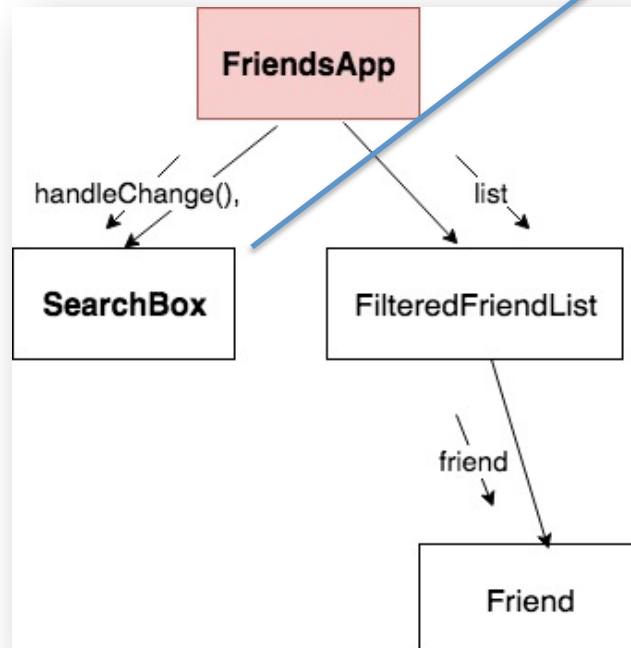
# Sample App – Architecture..



DEMO

# Inverse data flow

- What if a component's state is effected by an event in a subordinate component?
- **Solution:** The inverse data flow pattern.



## Friends List

- **Joe Bloggs**  
[jbloggs@here.com](mailto:jbloggs@here.com)
- **Paula Smith**  
[psmith@here.com](mailto:psmith@here.com)
- **Catherine Dwyer**  
[cdwyer@here.com](mailto:cdwyer@here.com)
- **Paul Briggs**  
[pbriggs@here.com](mailto:pbriggs@here.com)

. . . . . back to Lifecycle methods . . . .

# Sample App – Execution trail (Mounting & setState)..

The screenshot shows a web browser at `localhost:3000` displaying a web application titled "Friends List". The application has a search input field and a list of four friends: Joe Bloggs, Paula Smith, Catherine Dwyer, and Paul Briggs. Each friend's name is followed by a blue email link. The browser's developer tools are open to the "Console" tab, showing a sequence of log messages. A red circle highlights the `componentDidMount` message for `FriendsApp`. The log messages are as follows:

- render of FriendsApp
- render of FilteredFriendList
- componentDidMount of FriendsApp
- render of FriendsApp
- render of FilteredFriendList
- render of Friend (Joe Bloggs)
- render of Friend (Paula Smith)
- render of Friend (Catherine Dwyer)
- render of Friend (Paul Briggs)

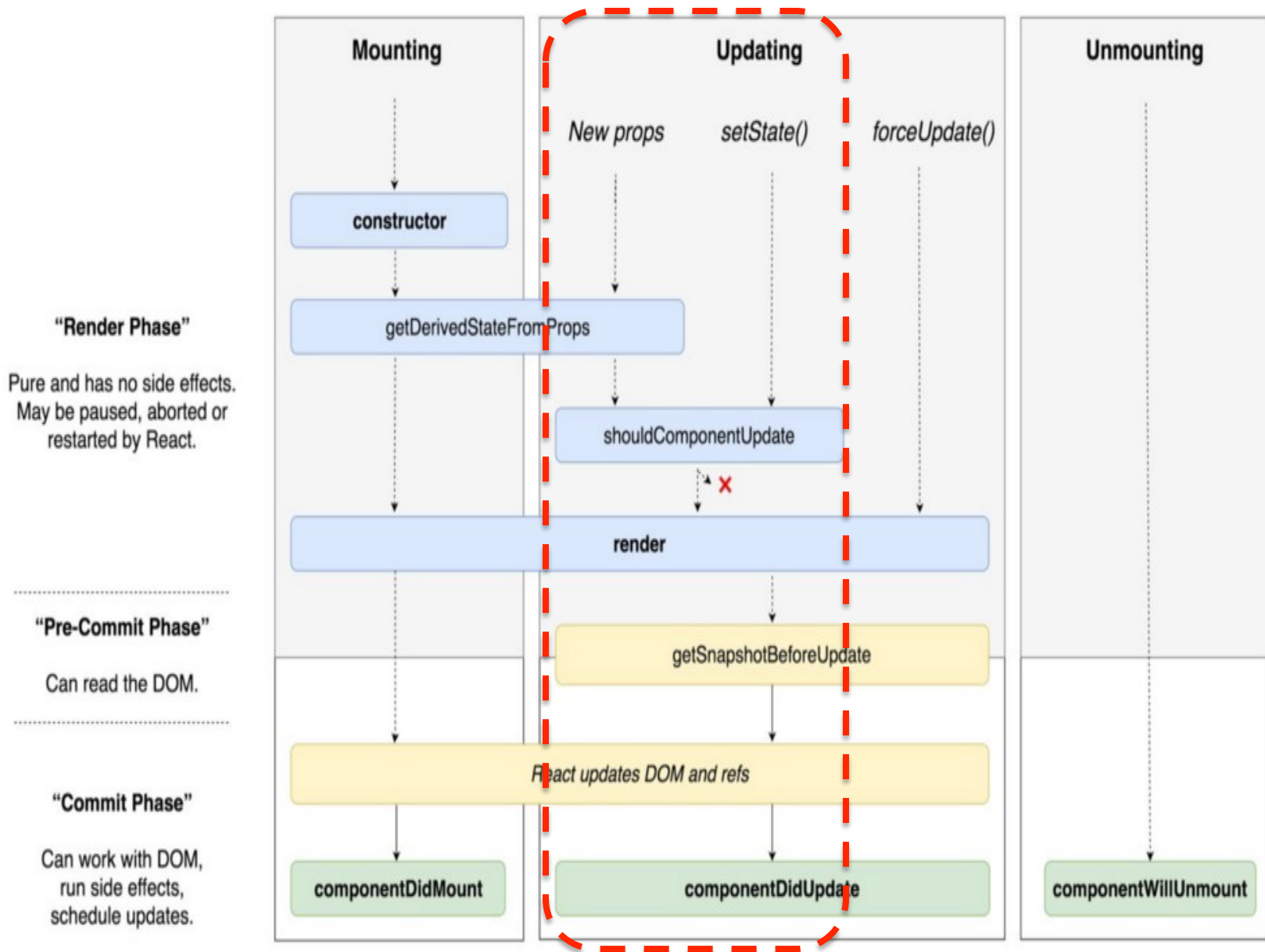
The application interface includes a search bar and a list of friends with their names and email addresses:

**Friends List**

Search

- **Joe Bloggs**  
[jbloggs@here.con](mailto:jbloggs@here.con)
- **Paula Smith**  
[psmith@here.con](mailto:psmith@here.con)
- **Catherine Dwyer**  
[cdwyer@here.con](mailto:cdwyer@here.con)
- **Paul Briggs**  
[pbriggs@here.con](mailto:pbriggs@here.con)





# Sample App – Execution trail (Update on new props & setState)..

## Friends List

- **Paula Smith**  
[psmith@here.com](mailto:psmith@here.com)

Note: The text box event handler calls setState() on FriendsApp

Elements Console Sources Network

top Filter

Console was cleared

< undefined

render of FriendsApp

render of FilteredFriendList

render of Friend (Paula Smith)

render of Friend (Paul Briggs)

render of FriendsApp

render of FilteredFriendList

render of Friend (Paula Smith)

render of Friend (Paul Briggs)

render of FriendsApp

render of FilteredFriendList

render of Friend (Paula Smith)

render of Friend (Paul Briggs)

render of FriendsApp

render of FilteredFriendList

render of Friend (Paula Smith)

render of Friend (Paul Briggs)

render of FriendsApp

render of FilteredFriendList

render of Friend (Paula Smith)

>

# Unidirectional data flow & Re-rendering

- **What happens when user types in to text box?**

***User types a character into text box***

***→ onChange event handler executes***

***→ Handler calls setState() (FriendsApp component)***

***→ React calls FriendsApp render() method***

***→ React calls render() method of children (FilteredFriendList) with new prop values***

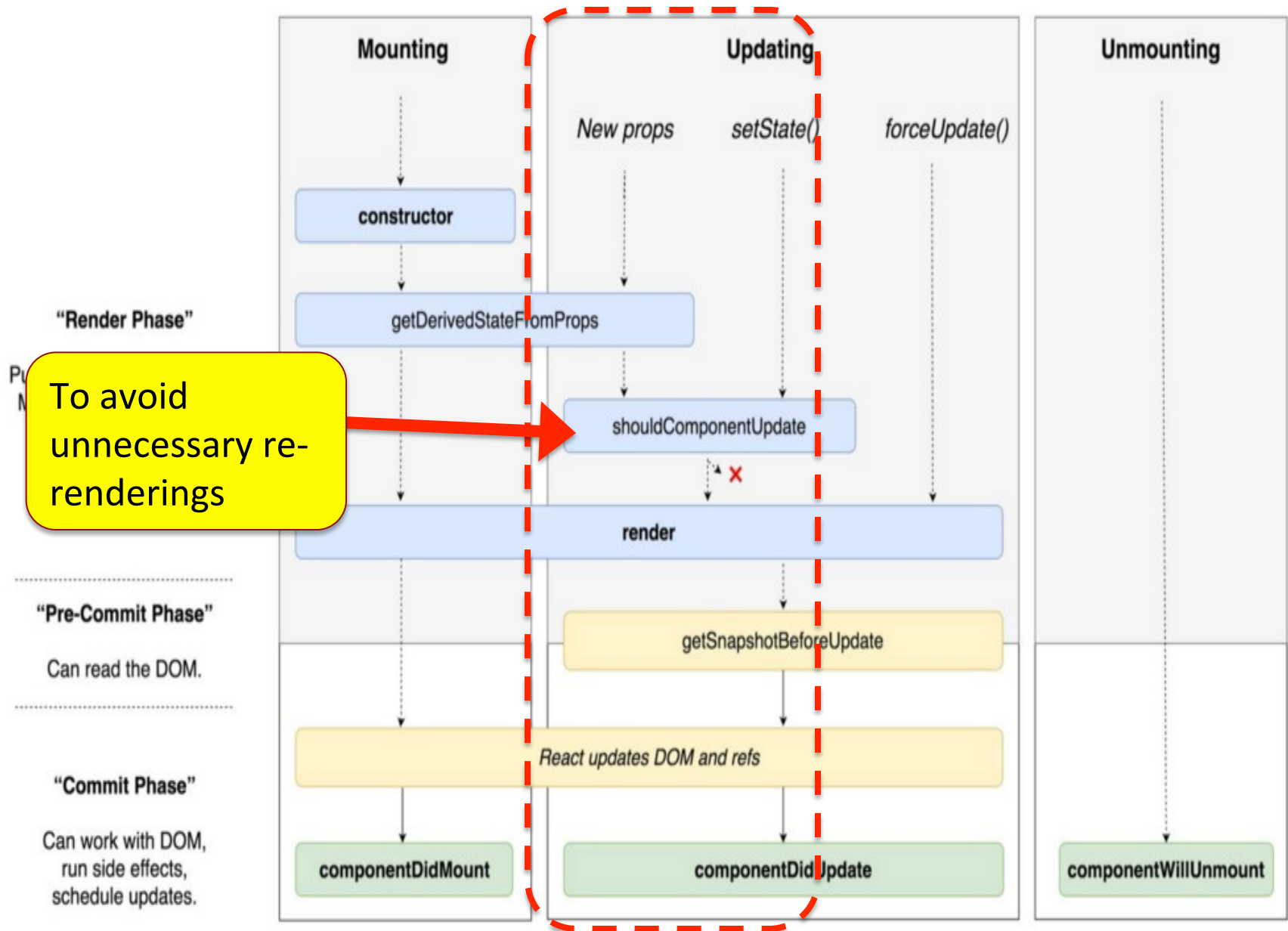
***→ React calls render() method of FilteredFriendList children.***

***→ (Pre-commit phase) React re-computes the new Virtual DOM***

***→ React diffs the new and previous Virtual DOMs***

***→ (Commit phase) React batch updates the Real DOM***

***→ Browser repaints screen***



# Sample App – Execution trail (Update on new props & setState)..

FilteredFriendsList should NOT re-render if the the length of array prop (of matching friends) has not changed

The screenshot displays a web application titled "Friends List" with a search input field containing the text "paula". Below the input, a list of friends is shown, including "Paula Smith" with a link "psmith@here.con". To the right, the browser's developer console is open, showing the "Console" tab. The console log shows the following sequence of events:

- Console was cleared
- undefined
- render of FriendsApp (indicated by a red arrow and the letter 'p')
- shouldComponentUpdate of FilteredFriendList
- render of FilteredFriendList
- render of Friend (Paula Smith)
- render of Friend (Paul Briggs)
- render of FriendsApp (indicated by a red arrow and the letter 'a')
- shouldComponentUpdate of FilteredFriendList
- render of FriendsApp (indicated by a red arrow and the letter 'u')
- shouldComponentUpdate of FilteredFriendList
- render of FriendsApp (indicated by a red arrow and the letter 'l')
- shouldComponentUpdate of FilteredFriendList
- render of FriendsApp (indicated by a red arrow and the letter 'a')
- shouldComponentUpdate of FilteredFriendList
- render of FilteredFriendList
- render of Friend (Paula Smith)

The red arrows and letters 'p', 'a', 'u', 'l', 'a' are used to highlight the sequence of renders and updates in the console log.

# Sample App – Execution trail (Update on new props & setState)..

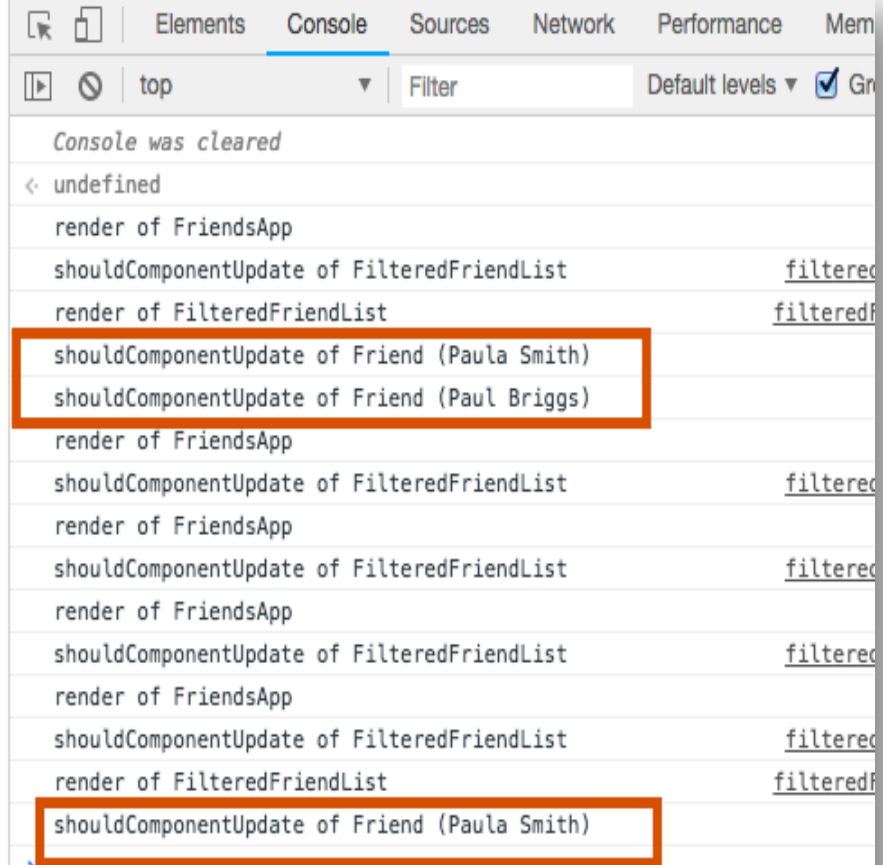
Friend should NOT re-render once it is mounted

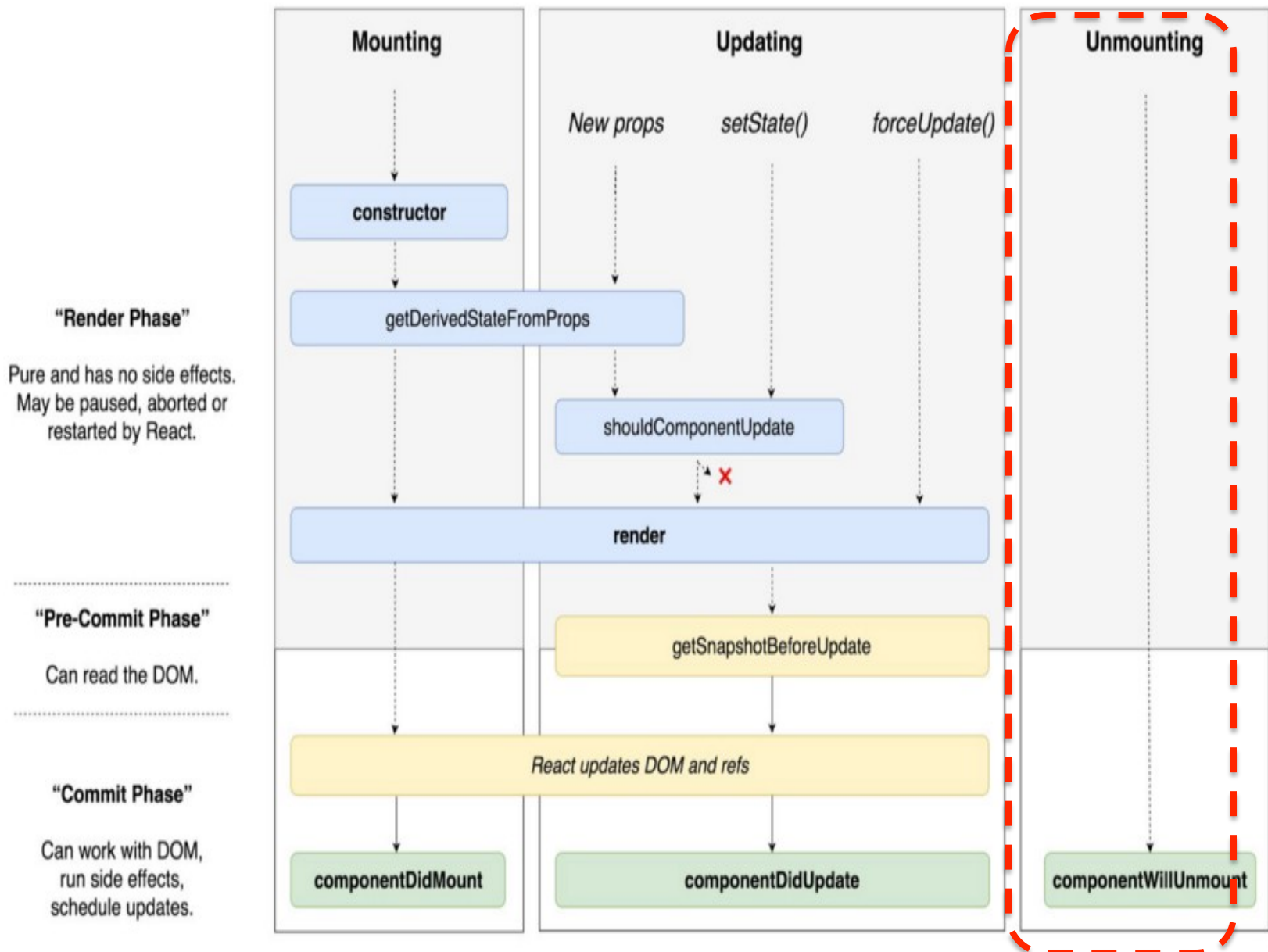
## Friends List

- **Paula Smith**

[psmith@here.com](mailto:psmith@here.com)

Note: All friends are mounted (and rendered) at app start-up.





# Sample App

- Execution trail (Un-mounting)..-

```
render of FriendsApp
render of FilteredFriendList
componentDidMount of FriendsApp
render of FriendsApp
shouldComponentUpdate of FilteredFriendList
render of FilteredFriendList
render of Friend (Joe Bloggs)
render of Friend (Paula Smith)
render of Friend (Catherine Dwyer)
render of Friend (Paul Briggs)
render of FriendsApp
shouldComponentUpdate of FilteredFriendList
render of FilteredFriendList
shouldComponentUpdate of Friend (Paula Smith)
shouldComponentUpdate of Friend (Paul Briggs)
componentWillUnmount of Friend (Joe Bloggs)
componentWillUnmount of Friend (Catherine Dwyer)
render of FriendsApp
shouldComponentUpdate of FilteredFriendList
render of FilteredFriendList
render of Friend (Joe Bloggs)
shouldComponentUpdate of Friend (Paula Smith)
render of Friend (Catherine Dwyer)
shouldComponentUpdate of Friend (Paul Briggs)
```

App start-up

Typed 'p'

Typed '<del>'



# Summary

- **For interactive apps, store the current 'state' of the UI in component(s) state object.**
- **React achieves DOM update performance improvements by managing a more efficient data structure, the Virtual DOM data structure**
- **A component's life spans from mounting to unmounting.**
- **A developer can link some logic in to the life span at preceribed times, using prescribed lifecycle methods.**